Java におけるオブジェクト中心のデバッガ

山﨑翔[†1]　久保田吉彦[†2]　紫合治[†2]

概要：プログラム開発に統合開発環境（以下，IDE） を用いた場合，デバッグ時には IDE の提供するデバッガを用いることが多い．ユーザは Graphical User Interface（以下，GUI）上でソースコード上にブレークポイントを設定し，デバッグ実行を行ない，ブレークポイントで実行を中断させプログラムの確認をする．しかし，ブレークポイントを適切な位置に設定するためには対象プログラムの動作に対する深い理解が必要となる．特に，オブジェクト指向言語によって構築されたシステムでは，ソースコード上の静的な場所だけでなく，実行時点での動的なオブジェクトの様子に沿ったデバッグが必要になるが，現在のデバッガではプログラマに対してオブジェクトの様子に沿った適切な GUI を提供できていない．本稿では実行時のオブジェクト全体の様子と個々のオブジェクトに着目した新しいブレークポイントを設定するための GUI を提案する．これは，Java プログラムの動作をアニメーションで表現するシステムを利用して，動的なオブジェクト図に対してブレークポイントを設定する GUI を備えたもので，Eclipse 上のプラグインとして実現されている．

キーワード：デバッグ，オブジェクト図，Eclipse プラグイン，オブジェクト中心のデバッグ

Object-Centric Debugger in Java

SHO YAMAZAKI[†1]　YOSHIHIKO KUBOTA[†2]
OSAMU SHIGO[†2]

Abstract: When programmer uses the integrated development environment (below, IDE) for program development, s/he uses the debugger provided by the IDE for debugging. Programmer sets the breakpoint on the program source code by using the Graphical User Interface (GUI), executes the program and checks the state of the program at the breakpoint. However, setting a breakpoint at the appropriate position of the program needs a deep understanding of the dynamic behavior of the target program. Especially for an object oriented program debugging, programmer should understand the dynamic behavior of the total objects and their states, and appropriate breakpoint may be at an object, not a class. The current debugger is not able to provide the appropriate GUI for such object oriented program debugging to the programmer. In this paper, we propose the plug-in tools that has a new breakpoint which focus on individual object and a GUI for setting breakpoint to object on the object diagram animator in Java programs.

Keywords: Debug Environment, Object Diagram, Eclipse Plug-in, Object-Centric Debugging

1. はじめに

プログラマは，プログラムを実行させ想定外のプログラム動作や変数の異常値を観測した際に，バグが混在している箇所を特定し，修正を行う必要がある．Sillito[1]らは，プログラマがソースコードの変更を行う際に抱く疑問を 44 種類定義している．プログラムの問題個所を特定する際には，それらの疑問を糸口にして，問題に関連していると思われる箇所にブレークポイントを設定し，ステップ実行で変数の値やオブジェクトの参照の変化を IDE の提供する GUI により確認する．しかしこのブレークポイントを適切な位置に設定するためには，ソースコード全体に対する深い理解が必要不可欠である．さらにオブジェクト指向言語によって構築されたシステムの場合，プログラムの実行はオブジェクト間の相互作用によって遂行されるため，UML やソースコードのような静的な表現では現せない時間と共に変化するシステムの状態をもとにした適切なブレークポイントの設定が必要になる．現在主流となっている実行スタックに焦点を当てたデバッガでは，オブジェクト指向言語に対するデバッグのための適切な UI を提供できていない．

この問題を解決するために Object-Centric Debugging[2]という手法が存在している．この手法は実行スタックではなく個々のオブジェクトに焦点を当てることでこの問題を解決するものである．ここでは，特定のオブジェクトのフィールドがアクセスされた場合とか，特定オブジェクトのメソッドが呼ばれたときなど，オブジェクトを指定したブレークポイントの設定を可能にする．しかしこの手法は現時点では Smalltalk を対象としたものであるため，Java ではまだ実現されていない．

本稿では，従来の実行スタックベースのデバッグ機能に加えて，プログラム実行時の動的なオブジェクトの様子をオブジェクト図として視覚的に表示することで，図中の特

†1 東京電機大学大学院情報環境学研究科
Graduate School of Information Environment, Tokyo Denki University
†2 東京電機大学情報環境学部
School of Information Environment, Tokyo Denki University

定のオブジェクトを指示しながら個々のオブジェクトに着目したデバッグ機能を提供するツールについて述べる．このツールは Eclipse の Java 開発環境のプラグインとして実現されており，Java プログラムの動的な振舞いを視覚化するためのプログラムアニメーション機能をもつ GUI をもとに，Object-Centric Debugging の機能を実装しており，オブジェクト指向プログラムの動作の理解しながらデバッグを進めることを可能にする．

以下に第２章で本研究の関連研究について述べ，第３章では本稿で提案するシステムの機能について説明する．第４章では本システムの構成について概要を述べ，さらに第５章では Eclipse デバッガと提案するシステムの違いについて説明する．最後に第６章で本稿のまとめと今後の課題について述べる．

2. 関連研究

Ressia[2]らはオブジェクト指向プログラムのデバッグでは，従来の実行スタック中心のデバッグではなく，個別のオブジェクトに着目した Object-Centric Debugging が必要であるとし，従来のソースコード中心のブレークポイントではないオブジェクト中心のブレークポイントを提唱した．オブジェクト中心のブレークポイントは，オブジェクトの状態に関するものとオブジェクトのインタラクションに関するものに分類できる．オブジェクトの状態に関するブレークポイントとしては次の二つを提唱している．

- Halt on write

いずれかのインスタンス変数もしくは指定のインスタンス変数に代入が行なわれた際に実行を中断する．

- Halt on read

いずれかのインスタンス変数もしくは指定のインスタンス変数が使用された場合中断する．

オブジェクトのインタラクションについては次の 6 つを提唱している．

- Halt on call

デバッグ対象のオブジェクトのメソッドが他のオブジェクトから呼び出された際，実行を中断する．メソッドは一つまたは複数指定できる．

- Halt on invoke

デバッグ対象のオブジェクトが他のオブジェクトのメソッドを呼び出したら実行を中断する．メソッドは一つまたは複数指定できる．

- Halt on creation

特定のクラスのインスタンスが生成されたとき実行を中断する．

- Halt on object in invoke

あるオブジェクトのメソッドが呼び出され，かつ特定のオブジェクトが引数として渡されたとき，実行を中断する．これは全てのオブジェクトが応答できる．全てのメソッドもしくはその一部に適用できる．

- Halt on object in call

特定のオブジェクトがメソッドの引数として使用されたら実行を中断する．全てのメソッド呼び出しもしくはその一部に適用できる．

- Halt on interaction

二つの特定のオブジェクトがお互いにメソッドの呼び出しをしていたら実行を中断する．

これらは Smalltalk で実装されており，オブジェクトを特定しコードを書くことによってオブジェクトにブレークポイントを設定する．但し，使いやすいインターフェースを提供するまでには至っていない．

Czyz[3]らは，Java の実行過程をオブジェクト図とシーケンス図によって視覚的に表現可能な Eclipse のプラグイン JIVE[4][5]を開発した．JIVE は過去のメソッド呼出は変数の変化を蓄積し，ブレークポイントで実行が中断された後，過去の変数の値やオブジェクト図やシーケンス図の観測が可能となっている．しかし，デバッグ機能は Eclipse に依存しており JIVE 独自のデバッグ機能はない．このため，Object-Centric Debugging のための機能は提供されていない．また JIVE では実行中のメソッド呼出の履歴や変数の値の変化の履歴をプログラム開始からすべて蓄積しているため，オブジェクトが多数現われるようなプログラムに適用すると，データの蓄積とオブジェクト図の描画に多大な時間がかかり，注目したいブレークポイントに到達するのに時間がかかりすぎるため，実際的なサイズのシステムのデバッグに対しては適用困難な場合がある．

我々は先行研究として Java プログラムの実行過程をオブジェクト図のアニメーションとして表現するシステムを開発した[6]．アニメーションの表現には UML の通常のオブジェクト図に実行中のメソッド表現や現在実行している個所を示すマークを追加した拡張オブジェクト図を使用している．プログラムの実行開始からアニメーションが始まり，オブジェクトのフィールドの変化や呼び出されるメソッドを強調表示しプログラムの終了までアニメーションを実行する．システムはマルチスレッドのプログラムにも対応し，アニメーション上では，スレッドはトークンと呼ぶ半透明の円形マークで表し，現在実行中のメソッド上に表示される．しかし，このシステムは IDE とは統合しておらず，オブジェクト指向プログラムの実行の様子の理解を助けるための独立したツールである．オブジェクト中心デバッグのためには，実行過程の理解支援に加えて，通常の IDE の機能，さらに Object-Centric debugging 機能までは統合したシステムが必要になる．

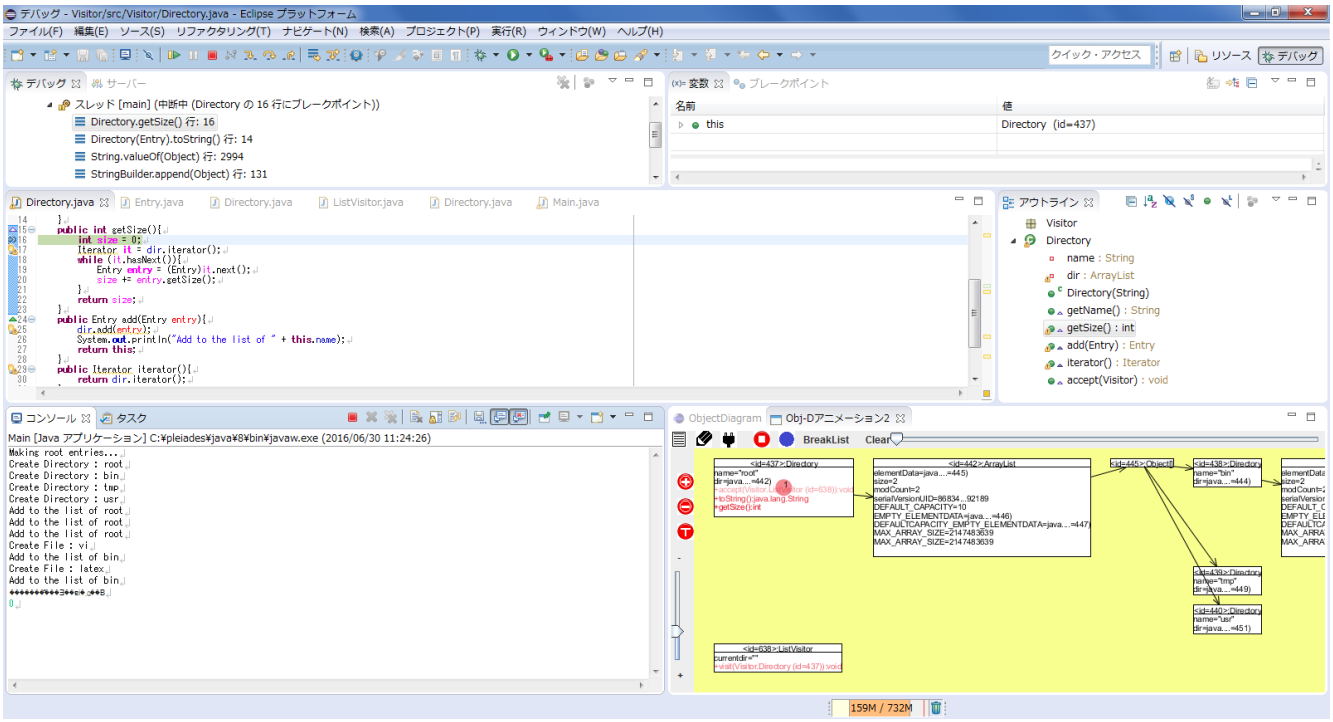

図 1　システムの全体図

3.　機能概要

本システムでは Eclipse の Java 開発環境のデバッガ機能の拡張として，Object-Centric Debugging の手法を取り入れたデバッグ機能とその機能を利用するための GUI を追加している．本章では我々が Object-Centric Debugging や既存のデバッガを参考にして新たに用意したオブジェクト中心のブレークポイントと，そのブレークポイントを設定するための GUI について述べる．

本システムは Eclipse デバッガを拡張する形で作られている．そのため我々が提供するデバッグ機能だけでなく従来の Eclipse デバッガの機能を併用することも可能である．拡張部分の中でも GUI 部分は一つのビューにまとめている．本プラグインを加えた Eclipse の全体図を図 1 に示す．

本システムを利用するにはまず，ソースコードの大まかな位置にブレークポイントを設定し，デバッグ実行を行う．ブレークポイントで実行が中断されたとき，図 1 右下のビューにその時点でのプログラムの状態を表したオブジェクト図が表示される．その後従来の Eclipse が提供しているステップ実行や実行再開に加えて，本システム独自のアニメーション実行の指示が行える．アニメーション実行では，プログラムを少しずつ実行しながら，プログラム内のオブジェクトの状態の変化に合わせてオブジェクト図がアニメーションで変化していくというものである．アニメーションを実行する速度を調整することでプログラムの実行速度をコントロールすることができる．また，アニメーションを一時停止することもでき，一時停止している間はプログラムの動作も中断される．さらに，アニメーションを注目したい特定の動作のところまで巻き戻して，その時のオブジェクトの様子を調べなおしたりすることもできる．但し巻き戻しはオブジェクト図の巻き戻しだけであり，プログラムの実行はアニメーションを止めた時点で中断している．また，アニメーション実行中での一時停止時点で本システムが提供するオブジェクト中心のブレークポイントの設定を行うことができる．このブレークポイントはアニメーション実行中のみ有効であるが，従来のブレークポイントと同時に使用することも可能である．オブジェクト中心の新しいブレークポイントについては以下 3.2 節で詳しく説明する．

3.1　アニメーションシステム

ブレークポイントの設定方法について説明する前に，アニメーションシステムについて触れておく．アニメーションシステムは Java プログラムの実行状態をオブジェクト図で表現し，各実行時点での状態の変化をアニメーションで表現することで，プログラムの動作を可視化するシステムである．アニメーションシステムでは従来のオブジェクト図に対して拡張を加えた拡張オブジェクト図を用いている（図 2，3）．以下にその変更点を示す．

- オブジェクト名の代わりに各オブジェクトが固有で持つオブジェクト ID を表示する．
- 二段目のセクションにそのオブジェクトがその時点で実行中のメソッドを表示し，文字の濃淡で実行順序を表す（濃い方が新しく実行されたメソッド）．

- メソッドの引数の型・名前の代わりにメソッド呼び出し時に渡された実引数を，基本データ型または文字列型の場合はその値を，それ以外のオブジェクトの場合はそのオブジェクトのオブジェクト ID を表示する．
- オブジェクト間の参照関係を参照側から被参照側への矢印で表現する．
- オブジェクトだけではなくスタティックなメソッドを持つクラスもオブジェクトと同様の形式で表示する．
- 現在処理が行っているメソッドのオブジェクトの位置を図 3 右のようなトークン（色つきの円）で表す．また，トークンの中にはスレッド ID を表示する．

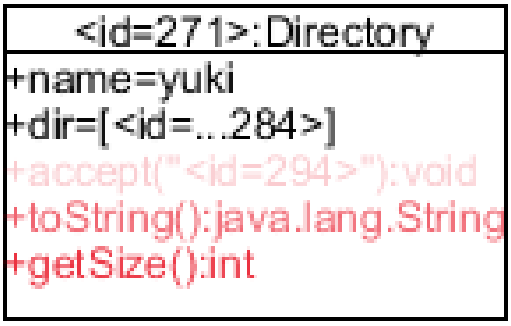

図 2　拡張オブジェクト図 1

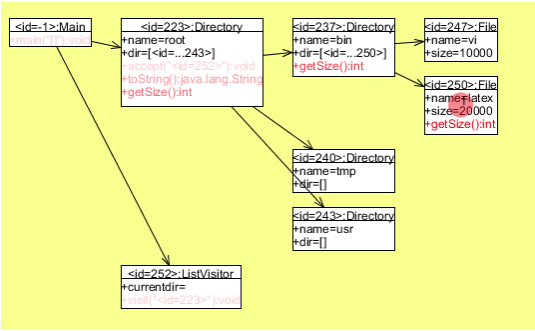

図 3　拡張オブジェクト図 2

ステップ実行を用いて変数の値やオブジェクトの参照の変化を観察する方法に比べて，それらを視覚的に分かりやすい形で表現している上にメソッドの実行順序などの情報も含まれているアニメーションシステムはプログラムの動作を理解するうえで役立つ．このアニメーションシステムはプログラマがブレークポイントを適切な位置に設定するために役立てることができるだけでなく，本システムで提案する新しいブレークポイントを設定するための UI として利用できるので本システムにも組み込まれている．

3.2 オブジェクト中心のブレークポイント

本システムにおけるオブジェクト中心のブレークポイントの設定方法を説明する．本システムで設定できるブレークポイントは大きく 2 つ，オブジェクトの状態に関連したものと，メソッドに関連したものの 2 つに分かれる．まず本システムの全体図及びオブジェクトの状態に関連するブレークポイントの設定を行う UI（図 4）及びメソッドに関連するブレークポイントの設定を行う UI（図 5）を示す．

オブジェクトの状態に関連するブレークポイントは一つの変数を対象としたものであり，その設定は変数タブで行う．ただし，配列に関してはブレークポイントを設定することができない．①はブレークポイントを設定する対象を表す．ここをクリックしてからオブジェクト図の中で設定したい変数をクリックする．スコープ（後述）をオブジェクトにした際は“変数名:オブジェクト ID”，クラスにした際は“変数名:クラス名”と表示される．②，③はそれぞれ対象へのアクセス・変更時に停止を行うかを設定する．片方のみだけでなくアクセス・変更の両方に対して停止するように設定することもできる．④はヒットカウント機能を使用するかの設定を行う．⑤でサスペンドポリシーの設定ができる．ここは VM とスレッドのどちらかに必ずチェックが入った状態になっている．⑥ではブレークポイントを設定するスコープを決める．オブジェクトの場合は一つのオブジェクトの変数を対象とし，クラスの場合はそのオブジェクトが属するクラスの全オブジェクトの変数を対象とする．⑦のボタンを押すと①～⑥の値がまとめられてブレークポイントとして設定される．その後，①～⑥の各項目は初期状態にリセットされる．

メソッドに関連するブレークポイントは一つのメソッド呼び出しを対象としたものであり，その設定はメソッドタブで行う．⑧は呼び出し元のオブジェクトまたはクラスを指定する．指定したオブジェクトまたはクラスのメソッドの中でメソッド呼び出しが発生した際に停止させることができる．クラスを指定した場合は指定したクラスの全オブジェクトのメソッド内でのメソッド呼び出し時に停止する．⑩は呼び出し先のどのオブジェクトまたはクラスのメソッド呼び出し時に停止させるかを指定する．クラスを指定した場合は指定したクラスの全オブジェクトのメソッド呼び出し時に停止する．⑫では特定のオブジェクトまたはクラスがメソッド呼び出しの引数として利用された際に停止するように設定する．クラスを指定した場合には指定したクラスの全オブジェクトのいずれかが利用されたときに停止する．⑧，⑩，⑫はそれぞれのラベルをクリックしてからオブジェクトの場合はオブジェクト図の中の設定したいオブジェクトを，クラスの場合は⑮のクラスリストから選択することで設定される．オブジェクトの場合は“<id=オブジェクト ID>”，クラスの場合はクラス名が表示される．これらの項目はその項目を指定せず，どのオブジェクトでも停止させるワイルドカードを設定できる．ワイルドカードの場合は“*****”と表示されている．⑨，⑪ではあらかじめ呼び出し元や呼び出し先のオブジェクトまたはクラスを指定したうえで，さらに特定のメソッドの中でメソッド

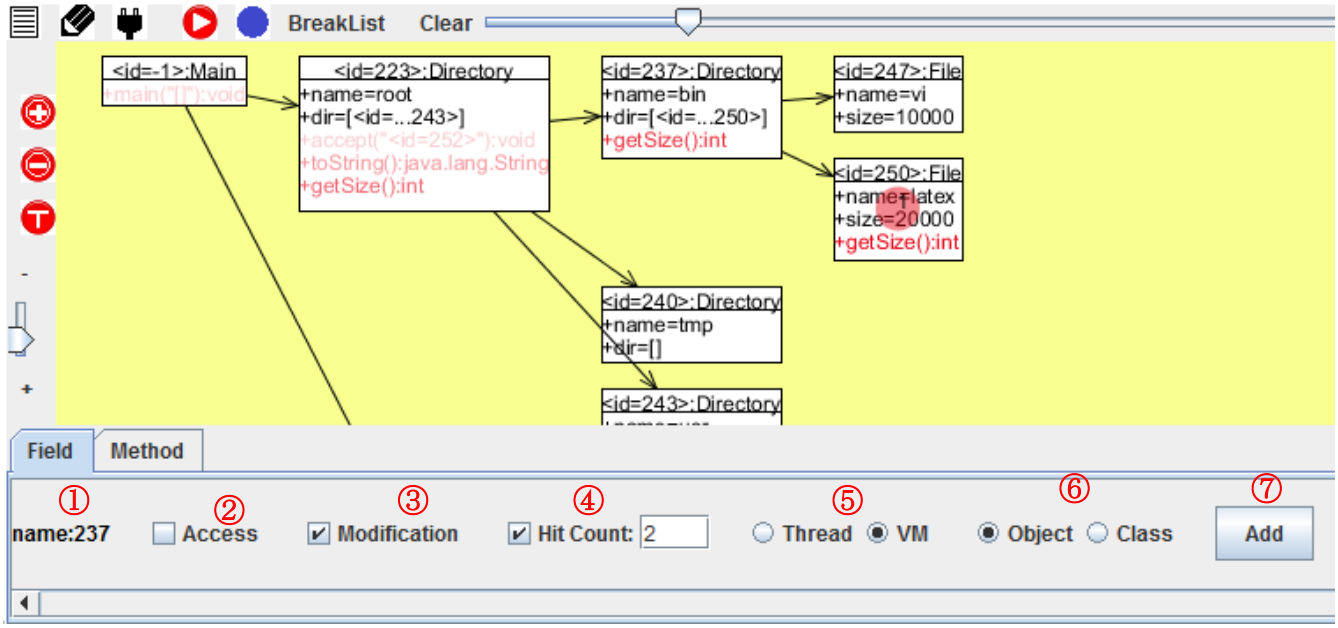

図 4　オブジェクト図と変数タブ

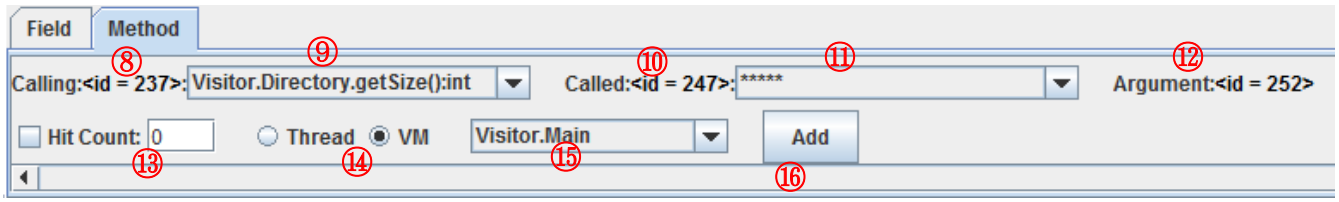

図 5　メソッドタブ

呼び出しが発生した際または特定のメソッドが呼び出された際に停止するように限定することができる．⑨，⑪は呼び出し元または呼び出し先に指定したオブジェクトまたはクラスが持つメソッドが格納されていて，そこから一つずつ設定することができる．⑬，⑭，⑯は変数タブにあるものと同様である．⑮はクラス一覧のリストである．クラスがロードされるとここに追加される．⑧，⑩，⑫でクラスを指定する際にはこれを用いる．

これらの設定項目を用いることで，関連研究で述べた Object-Centric Debugging[2]のブレークポイントのすべてを設定可能である．例えば，Halt on object in invoke を設定するには⑩と⑫を指定すればよい．ただし Halt on Interaction を設定するためには，二つのオブジェクトで呼び出し元と呼び出し先を入れ替えた二つのブレークポイントを設定する必要がある．

図 4 の上部にある breakList のラベルをクリックすると，変数・メソッドタブで追加したブレークポイントの一覧が表示される（図 6）．

リストから一つのブレークポイントを選択した状態で編集ボタンを押すと，変数またはメソッドタブにブレークポイントの情報が反映され，編集を行うことができる．また選択した状態で削除ボタンを押すとそのブレークポイントはリストから削除される．

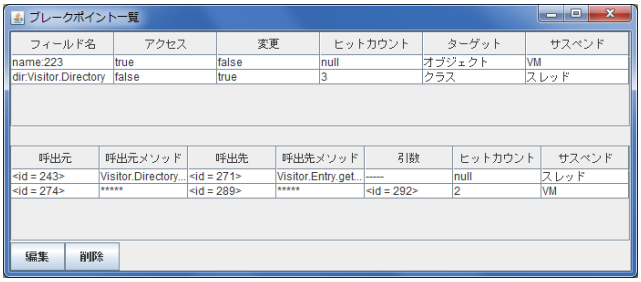

図 6　ブレークポイントリスト

4. 実現方式

4.1 全体構成

本システムはすべて Eclipse プラグインとして実装されている．その理由は Eclipse デバッガの機能を十分に利用するためのである．Eclipse ではすでに従来のブレークポイントやステップ実行といったデバッガとして必要な機能を利用可能であるうえ，Eclipse プラグインとして新しいデバッグ機能を追加する際にもそれらを利用することができるため開発に必要なコストを最小限にすることができる．

本システムはデータ抽出部とアニメータ・UI 部の二つに分かれる形で構成されている（図 7）．

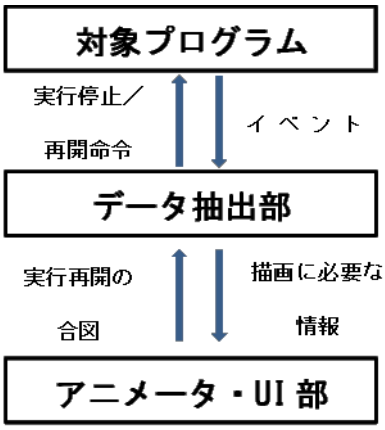

図 7 システム構成

データ抽出部では，最初にブレークポイントに到達した時点でオブジェクト図の描画に必要な情報を収集する．その後アニメーション実行中はメソッド開始・終了イベント発生時にデバッグ対象のプログラムの動作を一時停止し，アニメーションを描画するために必要な情報を収集する．そしてそれらの情報をアニメーション・UI 部へと送る．

アニメータ・UI 部ではデータ抽出部から送られてきた情報をもとに，オブジェクト図の描画・更新を行い，その後ブレークポイントの判定処理を行っている．ブレークポイントの判定は Java Debug Interface（以下，JDI）を利用しているものと，アニメータ・UI 部で行っているものの２種類がある（詳細については 4.3 節で述べる）．アニメータ・UI 部で判定を行っているものに関しては，ブレークポイントであると判定されない場合はデバッグ対象のプログラムの動作を再開させるために，実行再開の合図をデータ抽出部へ送る．データ抽出部がこの合図を受け取ることで実際に実行を再開させる．ブレークポイントであると判定された場合は，そのまま停止し，ユーザからの指示を待つ．

4.2 オブジェクトの収集と描画

Eclipse での Java プログラム開発は Java Development Tools(以下，JDT) が用いられており，JDIDebugTarget を拡張することでブレークポイント到達時やステップ実行の際に発生するイベントを取得できる．最初にブレークポイントに到達した際にオブジェクト図を作成する手順は次の通りである．

（1） 設定したブレークポイントにデバッグ実行が到達した際に発生するイベントを捕捉する．

（2） デバッグ対象の Java 仮想機械から，ユーザが設定したフィルタを介し必要なクラス群を取得する．

（3） 取得した各クラスのオブジェクトを全て取得する．取得には JDI の ReferenceType 型のオブジェクトへ instances(0) メソッドを使用する．取得後，id 番号をキーとして HashMap に保存する．

（4） 取得したオブジェクトを参照しているオブジェクトを再帰的に取得する．JDI の ObjectReference 型オブジェクトへ referringObjects(0) メソッドを再帰的に使用する．参照元が取得出来た場合，各参照元オブジェクトを id 番号をキーとして HashMap に保存する．

（5） 取得したオブジェクトの ID 番号と参照元オブジェクトの ID 番号からオブジェクト図を描画する．

アニメーション実行中のオブジェクト図の描画は以下の手順で行われる．

(1)メソッド開始･終了時に発生するイベントを補足する．

(2)イベントからメソッド名や引数などのメソッドに関する情報と，オブジェクト ID やクラス名・フィールドなどのオブジェクトに関する情報を取得する．

（3）オブジェクト ID をもとに図の変更を行うオブジェクトを特定し，オブジェクト図の更新を行う．

4.3 ブレークポイントの実装

本システムのブレークポイントの実装は JDI を利用している項目と，JDI にアニメータ・UI 部での処理を加えている項目の二つに分けることができる．

前者には変数タブの設定項目すべてとメソッドタブの設定項目のうちヒットカウント，サスペンドポリシー，呼び出し先のオブジェクトまたはクラスの指定の項目が該当する．フィールドのアクセス時及び変更時に停止させる場合，それぞれ com.sun.jdi.request パッケージの AccessWatchpointRequest と ModificationWatchpointRequest を用いる．ブレークポイントをオブジェクトに対して設定する場合は addInstanceFilter()をクラスに対して設定する場合は addClassFilter()を追加する．呼び出し先のオブジェクトまたはクラスの指定に関しては，MethodEntryRequest の addInstanceFilter()及び addClassFilter()を用いている．変数タブとメソッドタブに共通して，ヒットカウントは addCountFilter() を，サスペンドポリシーは setSuspendPolicy()を用いて設定している．

後者にはメソッドタブの設定項目の呼び出し元のオブジェクトまたはクラスの指定，呼び出し元のメソッドの指定，呼び出し先のメソッドの指定，引数の指定の項目が該当する．アニメーション実行中はアニメーションシステムを用いているため，MethodEntryEvent，MethodExitEvent が発生するたびに対象プログラムの実行を一時停止している．それらから情報を抽出し，あらかじめ GUI で設定したブレークポイントとの比較を行い，実行再開の指示を送るか停止したままにするかでブレークポイントの役割を果たしている．呼び出し先のメソッドと引数の指定に関しては MethodEntryEvent から実行するメソッド名や実引数を取得することが可能なので，取得したメソッド名や実引数とメソッドタブで設定したメソッド名や引数の比較を行い，ブレークポイントとして設定したメソッド引数か否かの判定を行う．呼び出し元のオブジェクトまたはクラスと呼び出

し元のメソッドに関しては，拡張オブジェクト図でメソッドの実行順序を表現するためにアニメータ・UI 部には実行中のメソッドに関する情報がスタックとして蓄えられているさ．このスタックを参照することによって呼び出し元のメソッドやオブジェクト・クラスを特定することができる．呼び出し元のメソッドやオブジェクト・クラスを特定することができれば，それらを変数タブやメソッドタブで設定したブレークポイントと比較を行うことで実行再開の合図を送るかを決定することができる．

5. Eclipse デバッガとの比較

Eclipse デバッガは JDI を用いて実装されている．ブレークポイントの設定やデバッグ実行・ステップイン・ステップオーバーを Eclipse の UI から行なうことが可能となっている．さらに，実行を一時停止する条件としてヒットカウントや条件式の指定などを設定することが可能である．

オブジェクト中心デバッグを Eclipse のデバッガによって実現する方法として，特定のインスタンスに限定してブレークポイントを有効にするインスタンス・ブレークポイントが考えられる．

オブジェクト中心デバッグを Eclipse のデバッガで実現するには，既存の UI 操作で可能なインスタンス・ブレークポイントの設置で可能なものと困難なものが存在する．

オブジェクト中心デバッグの内，フィールドの値の変化に反応する”Halt on write”と”Halt on read”は両方とも Eclipse のデバッガで実現可能である．“Halt on write”と”Halt on read”共にインスタンス・ブレークポイントを設置することで実現できる．ソースコード上のフィールドが宣言されている行に Eclipse の UI からブレークポイントを設置しデバッグ実行をする．実行が一時停止された際，インスタンス・ブレークポイントを対象のインスタンスに指定することで特定のインスタンスに限定して，変数へのアクセスや変更時に実行の一時停止をさせることが可能である．

オブジェクト中心デバッグの内，オブジェクトのインタラクションにおけるについて，Eclipse のデバッガから”Halt on call”と”Halt on create”が実現可能である．”Halt on call”は特定のオブジェクトのメソッドが呼び出された時に実行を一時停止する．Eclipse のデバッガからソースコード上のメソッドが宣言されている行にブレークポイントを設置し，デバッグ実行中にインスタンス・ブレークポイントを実行中断の対象となるインスタンスに指定することで実現することが可能である．”Halt on create”は指定したクラスのオブジェクトが生成されると実行を一時停止する．Eclipse でインスタンス生成時に実行を一時停止させるクラスのコンストラクタにブレークポイントを設置することで実現が可能である．

一方，Eclipse のデバッガでは実現が困難なオブジェクト中心デバッグが存在する．”Halt on invoke”は，呼び出されたメソッドとメソッドの呼び出し元を指定し実行を一時停止させる．メソッドの呼び出し元のオブジェクトは実行中のスタックフレームから直前のスタックフレームをたどることで見つけることが可能であるが，そのオブジェクトの指定を Eclipse のデバッガ上で指定し，実行を一時停止させるのは困難である． また，“Halt on object in invoke”はメソッドの呼び出し元と引数を指定し，実行を一時停止させる．直前のスタックフレームから指定された呼び出されるメソッドのオブジェクトであることと指定された引数であることを指示し条件を満せば実行を一時停止する必要があるため，Eclipse のデバッガでは操作が困難である．”Halt on object in call”はメソッドに条件付きブレークポイントを設置し，インスタンス・ブレークポイントを設置することで可能となる場合がある．

我々の提案するシステムでは Eclipse のデバッガでは煩雑な操作が必要なブレークポイントや Eclipse のデバッガでは困難な条件のブレークポイントを，オブジェクト図への指示によって直感的な操作で設置することが可能となっている．

図 8 は本システムの実行例である．Eclipse では困難である“Halt on invoke”は，本システムでは呼び出し元のオブジェクト<id=271>と呼び出されるメソッド<id=284>の getSize()をオブジェクト図から指定することで煩雑な操作をすることなく設定することが可能となっている．また，”Halt on object in invoke”への対応として<id=271>が<id=243>から accept()メソッドの引数として<id=294>のオブジェクトが渡された場合にも図から指示することで可能となっている．さらにワイルドカードを指定することによって<id=271>が他のオブジェクトのメソッド呼び出しを行なった場合にも実行を一時停止させるといった柔軟なブレークポイントの設定が可能である．

6. おわりに

本稿では，従来の実行スタックベースのデバッグ機能に加えて，プログラム実行時の動的なオブジェクトの様子をオブジェクト図として視覚的に表示することで，図中の特定のオブジェクトを指示しながら個々のオブジェクトに着目したデバッグ機能を提供するツールについて述べた．このツールは Eclipse の Java 開発環境のプラグインとして実現されており，Java プログラムの動的な振舞いを視覚化するためのプログラムアニメーション機能をもつ GUI をもとに，Object-Centric Debugging の機能を実装しており，オブジェクト指向プログラムの動作の理解しながらデバッグを進めることを可能にした．

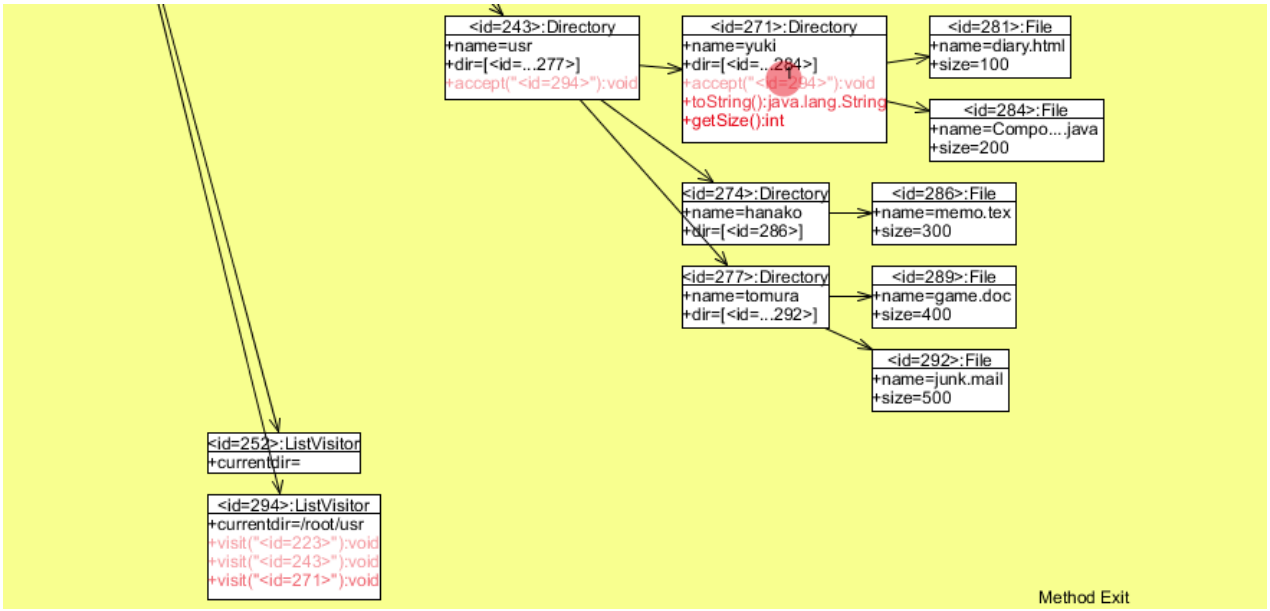

図 8　本システムの実行例

同種のオブジェクトが多数あり，その内の 1 つのオブジェクトに開発者の興味がある場合，例えば Visitor パターンを利用したプログラムのデバッグで Element に相当する多数あるオブジェクトの 1 つの状態や振舞いに異常がある為，デバッグを行なうとする．従来のデバッガではソースコードにブレークポイントを設置する為，多数のオブジェクトが存在することで頻繁にブレークポイントに到達し，そのたびに実行が停止することになる．我々の提案するデバッグ手法ではオブジェクト全体を俯瞰できるオブジェクト図を観察し，興味のあるオブジェクトにのみブレークポイントを設置することが可能であり，そのオブジェクトのブレークポイントに到達した時のみ実行が停止する，このようなケースでは従来のデバッガより効果的なデバッグが可能である．

JIVE と違い，本システムでは Eclipse のデバッグ機能だけでなく独自のデバッグ機能を追加している．また，Ressia[2]らが提唱した 8 種類のブレークポイントのうち，Halt on Interaction 以外は本システムが提供する GUI で簡単に設定することができる．しかし Halt on Interaction に関してはメソッド呼び出しを片方ずつ，二つのブレークポイントを設定する必要がある．また，多くのプログラム視覚化システムは小規模なプログラムを対象としている．しかし，本システムは可視化する範囲を限定し，プログラム実行の情報を最初から最後まで集めないことで大規模なシステムにも対応している．

今後の展望としては，本システムの機能拡張と評価を行っていきたい．現在検討中の機能としてはブレークポイントの可視化，オブジェクト図の絞り込み，実行の巻戻しがある．ブレークポイントの可視化機能は，本システムで設定したブレークポイントを対象にそのブレークポイントがどのオブジェクトからどのオブジェクトへのブレークポイントなのかなどをアニメーションシステム上で表示する機能である．オブジェクト図の絞り込みはデバッグ時に問題個所と関係ないと判断したオブジェクトをオブジェクト図から取り除いていくことによってオブジェクト図で見える範囲を限定し，問題個所と考えられる範囲を絞り込んでいく機能である．実行の巻戻し機能は，オブジェクト図のアニメーションを巻き戻した際に，対象プログラムの実行をそれに合わせて巻き戻し，再実行を行うことができる機能である．これらの機能追加と合わせて，本システムの有効性や使いやすさについて評価を行っていきたい．

参考文献

[1] Sillito, Jonathan, Gail C. Murphy, Kris De Volder. "Questions programmers ask during software evolution tasks." *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2006.

[2] Ressia, Jorge, Alexandre Bergel, Oscar Nierstrasz. "Object-centric debugging." *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012.

[3] Czyz, Jeffrey K., Bharat Jayaraman. "Declarative and visual debugging in eclipse." *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*. ACM, 2007.

[4] Gestwicki, Paul, and Bharat Jayaraman. "Methodology and architecture of JIVE." Proceedings of the 2005 ACM symposium on Software visualization. ACM, 2005.

[5] JIVE: Java Interactive Visualization Environment http://www.cse.buffalo.edu/jive/

[6] 山﨑翔, 久保田吉彦, 紫合治. "オブジェクト図のアニメーション." *ソフトウェアエンジニアリングシンポジウム 2015 論文集* 2015 (2015): 129-136.

[7] 久保田吉彦, 山崎翔, 紫合治. "デバッグ環境にオブジェクト図を提示する Eclipse プラグインの開発." *研究報告ソフトウェア工学 (SE)* 2016.29 (2016): 1-8.